"Express Mail" Mailing Label No. EL 521 771 215 US
Date of Deposit June 1, 2001

Patent
Attorney's Docket No. 032481-021

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of	
Mukesh Patel	
Application No.: 09/687,777	Group Art Unit: unknown
Filed: October 13, 2000	Examiner: unknown
For: JAVA HARDWARE ACCELERATOR USING MICROCODE ENGINE	

PETITION TO MAKE SPECIAL UNDER 37 C.F.R. §1.102(d)

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

Applicant petitions to make the above-captioned application special as an accelerated examination under MPEP 708.22 (VIII) by making the following statements:

A pre-examination search has been made. The classes and subclasses searched include Class 709, Subclass 1; Class 712, Subclasses 200, 245; Class 717, Subclasses 5, 6, 7, 8 and 9.

Discussion of the References

Evoy, USPN 5,937,193, describes a circuit arrangement for translating platform independent instructions for executing on a hardware platform and method thereof. A translating circuit is coupled to a processor and memory of a computer system to translate platform independent instructions, such as JAVA bytecodes, into corresponding native instructions for execution by the processor.

Dickol et al., USPN 5,898,850, describes a method and system for executing a non-native mode-sensitive instruction within a computer system. The system converts non-ntive instructions into native instructions.

Gosling, USPN 5,668,999 is a software application that describes a verifier for use with programs using data-type-specific bytecodes for verifying the proper operation of the executable program prior to the actual operation by the host processor.

Gosling, USPN 5,748,964, describes a software program interpreter for computer program written in a bytecodes language uses a restricted set of data-type-specific bytecodes. The program interpreter checks to see whether any instructions violate predefined stack usage and data type usage restrictions.

"Sun says JAVA chips will vastly increase speed, reduce costs to run JAVA programs," describes a dedicated JAVA processor. The dedicated JAVA processor runs JAVA as native instructions.

Dinwiddie, Jr., et al., USPN 5,113,522, describes a dual processor system.

Yoshida, USPN 5,218,711, describes a system with a coprocessor with a microprocessor having program counter registers for the coprocessor.

Adams USPN 5,889,996, describes a software accelerator that works well with L1 processor cache integrated into central processing units.

Moore et al., USPN 5,809,336, describes a high-performance microprocessor system having a variable speed system clock.

Moore et al., USPN 5,784,584, describes a high-performance microprocessor using instructions that operate within instruction groups. An instruction-fetching unit fetches groups from the memory for use by the central processing unit.

Arunachalam, USPN 5,778,178, describes a software method and apparatus for enabling real-time bidirectional transactions on a network.

Shoji et al., USPN 5,764,908, describes a computer network system using program modules on different computers.

Moore et al., USPN 5,659,703, describes a microprocessor system with a hierarchical stack and method of operation. The system describes a method of implementing a push-down stack.

Tyma, USPN 5,903,761, describes a method of reducing the number of instructions in a computer program. This is a software way of making a computer program more efficient.

"Efficient JAVA VM Just-in-Time Compilation" by Andreas Krall describes a software just-in-time (JIT) compilation method used to speed up JAVA programs.

Wahbe et al., USPN 5,761,477, describes a method for the safe implementation of virtual machines. This method describes a software compilation of a JAVA program.

Tremblay et al., USPN 6,021,469, describes a hardware virtual machine instruction processor. This reference describes a dedicated JAVA processor that runs virtual machines directly as native instructions.

Levy, USPN 5,923,892, describes a host processor and coprocessor arrangement for processing platform independent code. This relies on using a stack system that includes a regular processor and a stack-based processor.

Evoy, USPN 5,937,193, describes a translating circuit coupled to a processor for converting JAVA bytecodes into corresponding native instructions for execution by the processor.

Evoy et al., USPN 5,953,741, describes a stack cache for a stack-based processor. The stack-based processor uses a stack cache for accelerating stack access operation.

O'Connor et al., USPN 6,026,485, describes an instruction folding for a stack-based machine. This reference describes an instruction decoder that allows the folding away of JAVA virtual machine instructions, pushing an operand onto the top of a stack as a precursor to a second JAVA virtual machine instruction that operates on the top of the stack operand. This reference describes an instruction decoder for a dedicated virtual machine hardware processor.

Levy, USPN 5,923,892, describes a host processor and stack based coprocessor arrangement for processing platform-independent code. The stack-based coprocessor uses a stack cache for accelerating stack access operations and accelerating the overall performance of the processor.

Chatter, USPN 5,838,165 describes high-performance, self-modifying, on-the-fly alterable FPGA architecture.

Nilsen et al., USPN 6,081,665, describes a software method for efficient executable portable bytecode computer programs.

Alexander III, et al., USPN 6,118,940, describes a software application for bench-marking bytecode sequences. This describes a benchmark method for evaluating just-in-time compilers and the like.

Dice et al., USPN 6,141,794, describes a software system for producing native code executable by a computer system, and describes a just-in-time compiler.

Stanly et al, USPN 4,524,416, describes a stack mechanism with the ability to dynamically alter the size of the stack in a data processing system.

Bothner, USPN 6,110,226, describes a JAVA development environment utilizing optimizing JAVA compiler.

Gosling, USPN 6,075,940, describes a software method of preverification of stack usage.

Rodriguez, USPN 6,139,199, describes a software just-in-time scheduler.

Halter, USPN 5,905,895, describes a method and system for optimizing non-native bytecodes. This reference describes a JAVA software optimization system.

Breternitz, Jr., et al., USPN 5,805,895, describes a method and apparatus for code translation of optimization.

Cierniak et al., USPN 6,131,191 describes software code implants for a compiler.

Tremblay et al., USPN 6,125,439, describes a process of executing a method on a stack-based processor. The stack-based processor implements the JAVA bytecodes as the native instruction.

Koppala et al., USPN 6,108,768, describes a reissue logic for reissuing instructions in a multi-issue stack-based computing system. This reference describes a JAVA-based processor.

Tremblay et al., USPN 6,065,108, describes a method of accelerating operating on non-quick instructions.

Fisk et al., USPN 4,587,612, describes a system to accelerate instruction mapping external to source and target instruction streams. The independent processor converting multi-field instructions for a CPU of a first kind for multi-field instructions for a CPU of a second kind without disrupting the logical flow or execution of source or target instruction streams.

Chilinski et al, USPN 4,631,663, describes a macro-instruction execution in a micro-program control processor.

Hopkins et al., USPN 4,763,255, describes a method for generating short form instructions in an optimizing compiler. This is a software method which is used to improve the quality of code generated by a compiler or assembly.

Li et al., USPN 4,783,738, describes an array processor with adaptive spatial-dependent and data-dependent processing capabilities.

Hopkins et al., USPN 4,961,141, describes a software compiler adapted to compile object code from source code in a manner that produces efficient object code for the computer.

Beckwith et al., USPN 5,136,696, describes a high-performance pipelined central processor for predicting the occurrence of execution of single-cycle instructions and multi-cycle instructions.

Driscoll et al., USPN 5,142,681, describes an APL-to-Fortran translator. It is an apparatus and method for translating computer programs from an array source language to a scalar target language.

Haigh et al., USPN 5,163,139, describes an instruction preprocessor for combining short memory instructions into virtual long instructions.

Hastings, USPN 5,193,180, describes a system for modifying relocatable object code files to monitor accesses to dynamically allocated memory. This is a software object code expansion program that inserts new instructions and data between pre-existing instructions and data of an object code file.

Daniel et al., USPN 5,201,056, describes a RISC microprocessor architecture with multi-bit tag extended instructions for selectively attaching a tag from either the instruction or input data to the arithmetic operation outport.

Kohn, USPN 5,241,636, describes a method of parallel instruction execution in a computer.

Bouchard et al., USPN 5,333,296, describes a pipelined CPU.

Hastings, USPN 5,335,344, describes a method of inserting new machine instructions into pre-existing machine code.

Eickemeyer et al., USPN 5,355,460, describes a digital computer system capable of processing two or more computer instructions in parallel.

Smith et al., USPN 5,430,862 describes the emulation of a CISC instruction by RISC instructions using two pipeline stages for overlapped CISC decoding and RISC instruction.

Richter et al., USPN 5,481,684 describes emulating operating system calls using an alternate instruction set.

Mooney et al., USPN 5,490,256, describes a method of calling 32-bit functions from 16-bit functions.

Hastings, USPN 5,535,329, describes a method and apparatus for modifying relocatable object code files and monitoring programs.

Blomgren, USPN 5,542,059, describes a dual instruction set processor having a pipeline functional stage that is relocatable in time and sequence order.

Scantlin, USPN 5,574,927, describes a RISC architecture computer configured for emulation of the instruction set of a target computer. Thus, the RISC architecture can emulate a target computer such as an Intel computer.

Isaman, USPN 5,692,170, describes an apparatus for detecting and executing traps in a superscalar processor.

Walters et al., USPN 5,768,593, describes a cross-compiler that converts non-native code into native code immediately prior to execution of the code.

Blomgren et al., USPN 5,781,750, describes a dual instruction set CPU able to with hidden software emulation mode.

Cragun et al., USPN 5,774,868, describes an automated sales promotion system using neural networks.

Dickol et al., USPN 5,898,885, describes a method and system for executing non-native stack-based instructions. This system includes a memory, instruction set converter and processor.

Coon et al, USPN 5,619,666, describes a method for extracting complex, variable-length computer instructions from a stream of complex instructions

Blomgren, USPN 5,634,118, describes a method wherein a stack-register swap or exchange instruction is executed by splitting the exchange into two halves.

Coon et al., USPN 5,983,334, describes a describes a method for extracting complex, variable-length computer instructions from a stream of complex instructions

Robertson, et al., USPN 6,209,077, describes a general-purpose programmable accelerator board.

Nomura, et al., USPN 4,860,191, describes a coprocessor with a data flow circuitry. The special purpose coprocessor is designed according to the data flow

architecture and executes a task according to a token prepared by the general-purpose processor, the token having sequence control information.

Cooper, et al., USPN 5,077,657, describes a coprocessor including an emulator assist unit which forms addresses of user instruction operands in response to emulator assist unit commands from the host processor.

Goettelmann, et al., USPN 5,313,614, describes a method and apparatus for direct conversion of programs in object code between different hardware architectures, and describes the translation between a source computer and a second target computer.

Goettelmann, et al., USPN 5,577,233, describes a method and apparatus for conversion of programs in object code between different hardware architectures.

Gafter, USPN 5,650,948, describes a method and system for implementing a software implementation with data-dependent conditions to a control flow graph with conditional expression, the system describing an algorithm for conditional branching statements.

Asghar, et al., USPN 5,794,068, describes a central processing unit (CPU) with a digital signall processing (DSP) preprocessor that converts instruction sequences intended to perform DSP functions and DSP function identifiers.

Debaere and Van Campenhout, "Interpretation and Instruction Path Coprocessing," The MIT Press (1990). This volume describes coprocessors.

Description of the Claims:

All of the claims include a hardware accelerator operably connected to the central processing unit, the hardware accelerator adapted to convert stack-based instructions into register-based instructions.

Claim 1 describes a reissue buffer adapted to store converted native instructions issued to the CPU along with an indication of the order of instruction. The system is such that when the CPU returns from interrupt, the buffer examines the indication to determine whether the reissue a stored native instruction value. None of the cited references discloses, suggests or gives a motivation for such a system. For this reason, Claim 1 is

believed to be allowable. Claims 1-9 are dependent upon Claim 1 and for that reason are believed to be allowable.

Claim 16 describes using a microcode stage, the microcode stage including a microcode memory, the microcode memory outputting a number of fields, the fields including a first set of fields corresponding to native instruction fields, and a control bit field that affects the interpretation of the first set of fields. None of the cited references describes a hardware accelerator with such a microcode stage. For that reason, Claim 16 is believed to be allowable. Claims 17-21 are dependent upon Claim 16 and for that reason are believed to be allowable.

Claim 22 describes a hardware accelerator with a microcode generator unit and microcode interpretation logic. None of the above references describes or gives a suggestion or motivation for such a system. For this reason, Claim 22 and dependent claims 23-26 are believed to be allowable.

Claim 27 describes a hardware accelerator storing an indication of a top-of-operand stack pointer. The top-of-operand stack pointer is updated in hardware wherein when more than one stack-based instruction is translated into a single register-based instruction. The top-of-operand stack pointer is modified to reflect the effects of the register-based instruction, the stack-based instruction, and the instruction level parallelism. Such a system is not disclosed, suggested or given a motivation for in the cited art. For this reason, Claim 27 and dependent claim 28 are believed to be allowable.

Claim 29 includes a hardware accelerator for storing an indication of the depth count of a portion of the operand stack in the CPU register file, the depth count being updated during the translation process.

None of the cited references discloses, suggests or gives a motivation for such a system. For this reason, Claim 29 is believed to be allowable. Claim 30 is dependent upon Claim 29 and for that reason is believed to be allowable.

Claim 31 is a system with a hardware accelerator using an indication of the depth count of the portion of the operand stack stored in the CPU's register file. A hardware accelerator generates load instructions to load operand stack data from external memory to the register file is the depth is below a minimum depth; if the depth is above the maximum

depth, the hardware accelerator generates store instructions to move operand stack data from the register file to the external memory. Such a system is not disclosed, suggested or given a motivation for in the cited references. For this reason, Claim 31 is believed to be allowable. Claims 32 and 61-62 are dependent upon Claim 31 and for that reason are believed to be allowable.

Claim 33 describes a system with a hardware accelerator storing an indication of operands and variables stored in the register file of the CPU, the stored indication being used in the conversion process and being updated by the hardware accelerator. Such a system is not disclosed, suggested or given a motivation for in the cited references. For this reason, Claim 33 is believed to be allowable. Claim 34 is dependent upon Claim 33 and for that reason believed to be allowable.

Claim 37 describes a system with a hardware accelerator wherein at least the top four entries of the operand stack stored in the native CPU register file as a ring buffer, the ring buffer maintained in the hardware accelerator and operably connected to an overflow/underflow unit. Such a system is not disclosed, suggested or given a motivation for in the cited references, and for this reason claim 37 is believed to be allowable. Claim 38 is dependent upon Claim 37 and for that reason believed to be allowable.

Claim 39 describes a system including a hardware accelerator storing JAVA variable in the native CPU register file and an indication of which variables are in the native CPU register file. Such a system is not disclosed, suggested or given a motivation for in the cited references, and for this reason Claim 39 is believed to be allowable. Claim 40 is dependent upon Claim 39 and for that reason believed to be allowable.

Claim 41 describes a system with a hardware accelerator, the hardware accelerator composing native instructions based on the availability of variables and operands in the native CPU register file. This system is not disclosed, suggested or given a motivation for in the cited references, and for this reason Claim 41 is believed to be allowable. Claim 42 is dependent upon Claim 41 and for that reason believed to be allowable.

Claim 43 describes a system with a hardware accelerator marking variables in the native CPU register file as modified when updated by the execution of the JAVA bytecode. Such a system is not disclosed, suggested or given a motivation for in the cited references,

and for this reason the claim is believed to be allowable. Claims 44-45 are dependent upon Claim 43 and for that reason believed to be allowable.

Claim 46 describes a system with a hardware accelerator issuing native load instructions when a variable is not present in the native CPU register file, the memory address being computed by an ALU in the hardware accelerator. This system is not disclosed, suggested or given a motivation for in the cited references, and for this reason the claim is believed to be allowable. Claim 47 is dependent upon Claim 46 and for that reason believed to be allowable.

Claim 48 describes a system with a hardware accelerator composing native instructions where the native instruction operands contain at least two native CPU register file references where the register file contents are data for the operand stack. Such a system is not disclosed, suggested or given a motivation for in any of the cited prior-art references, and for this reason Claim 48 is believed to be allowable. Claim 49 is dependent upon Claim 48 and for that reason believed to be allowable.

Claim 50 describes a system with a hardware accelerator generating a new JAVA PC due to a GOTO or a GOTO_W bytecode. The system of Claim 50 is not disclosed, suggested or given a motivation for in the cited references, and for this reason Claim 50 is believed to be allowable. Claim 51 is dependent upon Claim 50 and for that reason believed to be allowable.

Claim 52 describes a system with a hardware accelerator generating new JAVA PC due to a JSR or a JSR_W bytecode computing the return JAVA PC and pushing the return JAVA PC on the operand stack. This system is not disclosed, suggested or given a motivation for in the cited references, and for this reason Claim 52 is believed to be allowable. Claim 53 is dependent upon Claim 52 and for that reason believed to be allowable.

Claim 54 describes a system with a hardware accelerator sign extending the SiPush and BiPush bytecodes and appending them to the immediately filed native instruction being composed. This system is not disclosed, suggested or given a motivation for in the cited prior-art references. For this reason Claim 54 is believed to be allowable. Claim 55 is dependent upon Claim 54 and for that reason believed to be allowable.

Claim 56 describes a system including a hardware accelerator that sign extends the SiPush and BiPush instructions and makes them available to be read by the native CPU. There is no disclosure or suggestion of such a system in the cited references. For this reason Claim 56 is believed to be allowable. Claim 57 is dependent upon Claim 56 and for that reason believed to be allowable.

Claim 58 describes a system with a hardware accelerator adapted to increment the JAVA PC within the hardware accelerator by generating an increment value based on the number of bytecodes being disposed, the JAVA PC incremented in the correct manner if multiple bytecodes are disposed at the same time. Such a system is not disclosed, suggested or given a motivation for in the cited references. For this reason Claim 58 is believed to be allowable. Claim 59 believed to be allowable because it is dependent upon Claim 58.

Enclosed with the Petition is the fee required under 37 C.F.R. §1.17(h).

The Applicant requests that this Petition to Make Special be approved.

Respectfully submitted,

BURNS, DOANE, SWECKER & MATHIS, L.L.P.

By: 76m.2

Kirk M. Nuzum
Registration No. 38,983
Redwood Shores, California Office
Tel: (650) 622-2300

P.O. Box 1404
Alexandria, Virginia 22313-1404

Date: June 1, 2001